VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Adam Gaura

Bakalářská práce

Vedoucí práce: RNDr. Eliška Ochodková, Ph.D.

Ostrava, 2021

Abstrakt

Cílem této bakalářské práce je popsat mé působení, přínos a získané znalosti ve společnosti Y Soft Corporation, a.s. Během praxe jsem se věnoval testování softwaru SafeQ v ostravském týmu. Za pomocí programovacího jazyka C#, frameworku SpecFlow a knihovny Selenium jsem vytvářel sady testů pro software do tiskáren SafeQ, který je mimo jiné nasazený i na VŠB. Dále jsem se věnoval regresním testům robotů.

Klíčová slova

automatizace, testování, praxe, C#, SpecFlow framework, Y Soft Corporation a.s., bakalářská práce

Abstract

The goal of this bachelor thesis is to delineate my performance, contribution and acquired knowledge in Y Soft Corporation, a.s. During my practice, I focused on testing of SafeQ software within the Ostrava team. Using the programming language C#, framework SpecFlow and Selenium libraries I tested software for printers called SafeQ, which is also deployed at VŠB. Furthermore, I worked on regression tests for robots.

Keywords

automation, testing, practice, C#, SpecFlow framework, Y Soft Corporation a.s., bachelor thesis

Poděkování

Rád bych poděkoval firmě Y Soft Corporation, a.s. za možnost absolvování odborné praxe a mému konzultantovi Bc. Ondřeji Ruszovi za cenné rady, které mi během praxe předal. Děkuji také mé vedoucí práce, paní RNDr. Elišce Ochodkové, Ph.D. za poskytnutí důležitých informací a připomínek, bez kterých by tato práce nebyla úplná.

Obsah

Seznam použitých zkratek a symbolů

QA	–	Quality Assurance
RQA	–	Robotic Quality Assurance
UI	–	User Interface
ETUI	–	Embedded terminal user interface

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem mé bakalářské práce bylo provést praxi ve společnosti Y Soft Corporation, a.s., která se specializuje na správu podnikového tisku a pokročilou digitalizaci dokumentů a workflow pomocí svého softwaru s názvem YSoft SafeQ. Důvodem, proč jsem se rozhodl pro bakalářskou praxi, bylo získání znalostí o tom, jak vývoj softwaru probíhá ve velké skupině, a zjištění, co to znamená být jeho součástí. Dalším důvodem bylo uplatnění teoretických dovedností získaných během studia na univerzitě v praxi, a tím být přínosný pro společnost.

Byl jsem přijat na základě pohovoru, který ověřoval mé znalosti a způsobilost k provedení práce, která mi byla později přidělena. Byl jsem součástí ostavského týmu, který se zabývá průběžnou kvalitou vyvíjeného produktu (Software YSoft SafeQ).

Tato závěrečná práce se skládá z šesti kapitol. V další kapitole popíšu společnost a tým, se kterým jsem pracoval. Následující kapitola pojednává o úkolech, které jsem během praxe musel zvládnout. Čtvrtá kapitola představuje použitý software a technologie, programovací jazyky, rámce, vlastní nástroje vytvořené společností a software třetích stran. V předposlední kapitole popíšu svůj pracovní postup a problémy, které jsem musel překonat během návrhu a implementací automatizovaných testů. Nakonec shrnuji a hodnotím mou praxi, získané znalosti a nové dovednosti.

Kapitola 2

Y Soft Corporation, a.s.

2.1 O společnosti

Y Soft Corporation, a.s.[1] je společnost zabývající se vývojem softwaru a hardwaru. Byla založená v roce 2000 v Brně. V současné době působí v 17 zemích po celém světě s více než 300 zaměstnanci. Y Soft má více než 14 000 zákazníků a její řešení se používají ve 121 zemích na 6 kontinentech. Hlavním zaměřením společnosti je snižování nákladů souvisejících s tiskem, zvyšování efektivity a zdokonalování bezpečnosti. YSoft SafeQ je specializovaný software určený k dosažení tohoto cíle, který je určen středním a velkým podnikům. Společnost Y Soft také vyvíjí svůj vlastní hardware používaný k vylepšení tiskáren o další funkce. Mezi tato řešení patří USB čtečka karet a terminál SafeQ.

2.2 Tým

Byl jsem zařazen do ostravského oddělení tiskového managmentu, konkrétně do týmu "Smog", který má na starosti terminály na specifických vendorech a integrační testy. Má pracovní pozice v tomto týmu byla Quality Assurance. QA dohlíží na vývoj a zlepšování kvality celého produktu SafeQ. Tento tým s 7 členy je největším týmem ostravské pobočky Y Soft. Pracovní postup týmu je organizován do 14denních sprintů. V tomto časovém rozpětí jsou členové připraveni dokončit předem stanovené množství práce. Každý den se koná stand-up schůzka, aby všichni byli informováni o aktuálním postupu zadaných úkolů. Na konci každého sprintu má tým schůzku, aby se ohlédl za posledním sprintem a naplánoval další.

[1] Web společnosti: `https://www.ysoft.com/cs`

Kapitola 3

Úkoly zadané během praxe

Mým úkolem bylo navrhnout sadu testů, implementovat je a takto rozšířit testovací platformu pro software SafeQ. Během praxe jsem vytvářel sady testů pro následující uživatelská rozhraní: **Zobrazení tiskových úloh Waiting – Printed – Favorite (Záložka ve stavu vytištěné), Zobrazení tiskových úloh All-in-one, Zobrazení skenovacích možností uživatele, Zobrazení pokročilých skenovacích možností uživatele**. Všechny tyto testy byly zautomatizovány a při každé změně „master branch“ proběhné celá sada testů.

Master branch je hlavní větev v repozitáři, ve které se obvykle nachází poslední změny v kódu. Přímé úpravy v této větvi jsou zpravidla zakázány a nové změny se do této větve dostávají pouze skrze Pull requesty. Pull request je požadavek, díky kterému lze provést synchronizaci dvou větví, zpravidla to bývá synchronizace hlavní vývojové větvě s větví, která z ní přímo vychází. Tyto pull requesty jsou revidovány zkušenějším vývojářem, který je buď schválí nebo zamítne.

Všechny mnou v průběhu praxe vytvořené a implementované testy spolu s časem jejich realizace jsou uvedeny v tabulce 3.1

Uživatelská rozhraní	Počet testů	Čas realizace(h)
Zobrazení tiskových úloh All-in-one	31	102
Zobrazení skenovacích možností uživatele	26	140
Zobrazení pokročilých skenovacích možností uživatele	13	110
Zobrazení tiskových úloh	26	75
Waiting – Printed – Favorite: Záložka ve stavu vytištěné		

Tabulka 3.1: Testy vytvořené a implementované během praxe

3.1 Seznámení s produktem a prostředím

Po nástupu do firmy jsem prošel e-learningem, díky kterému jsem se seznámil s produkty a fungováním celé firmy. Pro lepší seznámení s produktem a jeho testováním mi byly přiděleny:

- **Manuální testy**

  Tyto testy jsou prováděny fyzickou osobou dle předem daného scénáře, kde jsou určené podmínky, kroky a očekávaný výsledek. Všechny testy se provádí na terminálu, kde je zobrazena komponenta ETUI, která zajišťuje správné zobrazení přístupového terminálu. Celý průběh zaznamenává v TestLinku (nástroji pro správu a organizaci testování softwaru) osoba, která testuje produkt.

- **Systémové testy pomocí robotické ruky**

  Po každé aktualizaci produktu přijde řada na tyto testy, které mají ověřit funkčnost/nefunkčnost produktu. Jsou důležité pro zajištění dosavadní kvality produktu. Před spuštěním testu je nutné zkontrolovat, zda hardware jak robota, tak i tiskárny je správně připojený a zkontrolovat, zda někdo tento hardware nepoužívá. Dále pak pomocí webové aplikace je zapotřebí zjistit, zda jsou robot a kamera zkalibrováni, a pokud nejsou, tak je zkalibrovat. Díky této webové aplikaci tester zjistí všechny potřebné informace k připojení a následnému spuštění testů. V neposlední řadě aplikace poskytuje možnost připojit se ke kameře robota a vidět, jak robot vykonává testy, a díky tomu také zaznamenávat celý průběh testů. Když jsem prováděl systémové testování pomocí robota, zjistil jsem, že ve dvou testech robot nezakliknul tiskovou úlohu, protože klikal těsně vedle. Obě chyby byly způsobené nesprávnou konfigurací robota, a proto oba testy selhaly. Tyto poznatky jsem předal kolegům z RQA.

- **Automatické testy robota**

  Tyto testy provádějí roboti na terminálech díky Robot Frameworku, který je na základě klíčových slov rozšiřitelný automatizační framework pro robotickou automatizaci procesu. Díky webovému uživatelskému rozhraní RMS (Robotic Management System) jde definovat zobrazené stránky na terminálu a rozmístění tlačítek, kalibrovat kameru robotů, spravovat zpětnou vazbu, posílat data, z kterých se AI naučí zpracování obrazů, a také prohlédnout poskytnuté statistiky.

3.2 Automatizované testy

Mým hlavním úkolem bylo vytvořit automatické UI testy pro terminálovou část YSoft SafeQ. Hlavním cílem automatizovaných testu je časová úspora. Takto vytvořené testy se automaticky spustí při jakékoli úpravě softwaru, a tímto se zajistí, že nové změny nijak neomezí funkčnost produktu. Před návrhem scénářů jsem musel stáhnout a nastavit veškeré potřebné nástroje a frameworky. Následně jsem se musel s těmito nástroji a frameworky dobře seznámit, hlavně s Gherkin a SpecFlow.

Všechny mé úkoly souvisely s automatizací integračních testů. Tyto testy byly vytvořeny pro komponenty Terminal Server a jejich integraci s komponentou ETUI, která zajišťuje správné zobrazení přístupového terminálu. Každému uživatelskému rozhraní, které jsem měl otestovat, byl vy-

tvořen task (úkol) v Jiře (více o tomto softwaru v kapitole Použité technologie: 4.6.3) a také vlastní větev, ve které následně probíhalo vytváření testů. Každý task má stručný popis, který zahrnuje charakteristiky úkolu, jeho cíl a celý průběh vývoje. Pro správu tasků Y Soft Corporation, a.s. využívá produktů firmy Atlassian, které umožňují jednotlivým úkolům přiřadit odkaz na BitBucket. Ve vytvořených odkazech jsou kódy následně kontrolovány. Dalším užitečným nástrojem používaným v této firmě je aplikace Sourcetree, která přehledně zobrazuje úpravy kódu a díky tomu umožňuje jejich kontrolu před implementací kódu do větve. Pro každé uživatelské rozhraní bylo nutné vytvořit vlastní sadu testů, aby byla zachována přehlednost kódu a jejich voláním otestována konkrétní část softwaru. Každá sada testů měla ověřit funkcionalitu uživatelského rozhraní z pohledu zákazníka.

Kapitola 4

Použité technologie

4.1 Platforma YSoft SafeQ

YSoft SafeQ [1] nabízí efektivní správu tiskových front a také pohodlné definování zásad a pravidel pro tisk. Tím efektivně uspokojuje požadavky jak podnikového, tak vzdělávacího sektoru, ohledně spolehlivé správy výstupů.

Toto jsou jeho hlavní funkce:

- **Ověřování**

  Umožňuje zabezpečený přístup k zařízením pro tisk, kopírování a skenování. Díky ověřování dokumenty zůstávají bezpečně uloženy, dokud se uživatel nepřihlásí. Možností přihlášení je několik: přiložením identifikační karty, kódem PIN, jménem/heslem nebo jejich kombinací.

- **Tisk na základě pravidel**

  Umožňuje nastavení pravidel pro tisk a kopírování, díky kterým se můžou snížit náklady. Například konverze barevných úloh na černobílé, tisk na obě strany papíru či přesměrování úlohy na úspornější zařízení.

- **Scan Workflows**

  Jedná se o šablony, které krok za krokem popisují pracovní postup skenování, definují základní nastavení, výstupní soubor a samotné parametry pracovního postupu. Tyto šablony usnadňují digitalizaci a distribuci dokumentů.

- **Kredit a účtování**

  Umožňuje klientovi vytvářet uživatelům virtuální kreditní účty, které pak můžou použít k platbě za tisk, skenování a kopírování podle rozpočtového střediska nebo klienta. Webová aplikace Cash Desk pak slouží k dobíjení kreditu.

- **Reporting**

  Centrálně sleduje náklady a využití tiskáren v celém tiskovém prostředí. Poskytuje skutečná data, která slouží k optimalizaci tiskových služeb. Může cílit na náklady celopodnikového tisku, různá oddělení nebo dokonce jednotlivé uživatele.

- **Print roaming/client based print roaming**

  Poskytuje bezpečný přístup k tištěným souborům z libovolné autorizované tiskárny v organizaci. Díky tomu jsou dokumenty přístupné z jakékoliv tiskárny v organizaci, která je připojená k síti. Úlohy jsou vytištěny až po ověření uživatele u zařízení, takže citlivé dokumenty nezůstávají bez dozoru ve výstupním zásobníku.

Na níže uvedeném obrázku 4.1 vidíme výše zmíněné funkce YSoft SafeQ zobrazené pomocí uživatelských workflow.

Obrázek 4.1: Uživatelská workflow YSoft SafeQ

4.2 Framework SpecFlow

SpecFlow [2] je open-source testovací framework, který podporuje metodiku vývoje řízeného požadavky na chování. Poskytuje mechanismy pro přeměnu specifikací na spustitelný kód. Tento kód lze poté použít k vytvoření automatizovaných testů, které běží při spuštěné aplikaci k ověření, zda skutečně vykazuje chování popsané v specifikacích. Scénáře psané pomocí syntaxe Gherkin, která nám umožňuje definovat a popsat chování systému lidsky čitelnou řečí, jsou uloženy v souboru s příponou *.feature*. Každý krok ze scénářů pak můžeme implementovat v jazycích C#, Java, Python, PHP atd. Já jsem implementoval kroky pomocí jazyku C#. Každé třídě přidáme atribut Binding a název souboru s definicí testů, aby překladač při kompilaci byl schopen spojit Gherkin definice s metodami v této třídě. Tyto metody se vykonají, pokud jsou zahrnuty v testovacím scénáři ve Feature souboru.

4.3 Gherkin

Gherkin [3] je jazyk, který vývojáři používají k definování testů. Tato syntaxe podporuje vývoj založený na chování softwaru. Jelikož tento jazyk používá čistou angličtinu, popisuje případy použití softwarového systému způsobem, který může číst a rozumět téměř kdokoli. Proto umožňuje vývojářům, manažerům, obchodním analytikům a dalším zúčastněným stranám porozumět požadavkům projektu. Díky tomuto jazyku je snadné vytvořit jednoduchou dokumentaci kódu, který se právě píše. Gherkin také poskytuje skripty pro automatizaci testů a podporuje desítky jazyků. Používá sadu speciálních klíčových slov, aby dal strukturu a význam spustitelným specifikacím. Každý řádek, který není prázdným řádkem, musí začínat klíčovým slovem, následovaným libovolným textem. Jedinou výjimkou jsou popisy funkcí a scénářů.

Klíčová slova jsou následující:

- **Feature**

  Gherkin dokumenty začínají tímto klíčovým slovem, následovaným textem s popisem. Jednoduše řečeno, feature je název a popis toho, co má testovat dané seskupení scénářů, které je uloženo v jednom zdrojovém kódu.

- **Descriptions**

  V případě potřeby lze pod tímto klíčovým slovem psát popisy ve volném formátu, pokud žádný z řádků nezačíná klíčovým slovem.

- **Background**

  Vytváří společný kontext všech scénářů. Používá se k tomu, aby před každým scénářem proběhlo nastavení, které je společné pro všechny scénáře, a tím se zamezilo zbytečnému opakování given kroků ve scénářích. Může obsahovat více klíčových slov given, ale pro každý soubor může existovat pouze jeden background.

- **Scenario**

  Uvozuje název a popis scénáře, je to klíčové slovo pro jeho začátek.

- **Scenario Outline**

  Lze použít ke spuštění stejného scénáře několikrát s různými kombinacemi hodnot. Parametry jsou odděleny pomocí těchto znaků: <>. Musí obsahovat část Examples. Kroky tohoto scénáře se nespustí nikdy přímo, slouží jenom jako šablona, která se spustí pro každý řádek v části Examples (kromě prvního řádku záhlaví). Ve výpisu zdrojových kódů 4.1 vidíme praktické využití scénáře s více vstupními parametry, kdy místo vytváření scénáře pro každý znak, vytvoříme jeden scénář s tabulkou parametrů.

- **Given**

  Popisuje předběžné podmínky a počáteční stav před zahájením testu, a umožňuje jakékoli nastavení softwaru, než s ním uživatel začne komunikovat. Část given tvoří předpoklady pro test.

- **When**

  Používají se k popisu události nebo akce. Může to být interakce mezi osobou a systémem nebo událost spuštěná jiným systémem.

- **Then**

  V tomto kroku je popsán očekávaný výsledek. Definice by měla použít tvrzení k porovnání skutečného výsledku (co systém ve skutečnosti dělá) s očekávaným výsledkem (to, co krok říká, že systém má dělat).

- **And**

  Lze připojit k jakémukoli typu kroku (Given, When, Then) a slouží jako logická konjunkce. Nemá žádný jedinečný význam – existuje jednoduše proto, aby byly scénáře čitelnější.

```
Feature: Calculator

    Scenario: Calculator should add two numbers
        Given The calculator is reset
            And I enter "7"
            And I enter "+"
            And I enter "5"
        When I click count
        Then The result should be "12"

    Scenario Outline: Should support
        Given The calculator is reset
            And I enter "<firstNumber>"
            And I enter "<inputOperator>"
            And I enter "<secondNumber>"
        When I click count
        Then The result should be "<newTotal>"
```

```
Examples:
  |firstNumber |inputOperator| secondNumber |newTotal|
  | 5          | +           | 7            | 12     |
  | 5          | -           | 7            | -2     |
  | 3          | *           | 4            | 12     |
  | 30         | /           | 3            | 10     |
```

Listing 4.1: Příklad scénářů psaných v jazyce Gherkin

Ve výpisu zdrojového kódu 4.2 vidíme implementaci pro každý krok z dříve zmíněného scénáře (výpis zdrojového kódu 4.1). Před každou implementací kroku je vždy v hranatých závorkách klíčové slovo, které určuje, o jaký krok se jedná (Given, When, Then). Dále pak v kulatých závorkách za znakem `@` je pokaždé v uvozovkách název kroku. Pokud je v názvu (.*), jedná se o krok se vstupem. Pak je již funkce, ve které je daný krok implementován, v jazyce C#.

```
public class CalculatorSteps
{
    private Calculator calculator;

    [Given(@"The calculator is reset")]
    public void GivenTheCalculatorIsReset()
    {
    calculator = new Calculator();
    }

    [When(@"I enter (.*)")]
    public void WhenIEnter(int character)
    {
        calculator.add(character);
    }

    [When(@"I click count")]
    public void WhenICount()
    {
        calculator.count();
    }
```

```
    [Then(@"The result should be (.*)")]
    public void ThenTheValueShouldBe(int result)
    {
        Assert.Equal(result,calculator.Value);
    }
}
```

Listing 4.2: Příklad souboru s definicemi kroků ze scénářů

4.4 Selenium

Selenium [4] je bezplatný (open-source) framework pro automatické testování, vyvinutý v programovacím jazyce Java. Používá se nejčastěji k testování webových aplikací napříč různými prohlížeči a platformami. K vytvoření testovacích skriptů Selenium se může použít více programovacích jazyků, jako je Java, C#, Python atd. Selenium není jen jeden nástroj, ale sada, skládající se z několika navzájem se doplňujících komponentů, přičemž každý kus vyhovuje různým testovacím potřebám.

Zde je seznam nástrojů:

1. **Selenium IDE**

   Nejjednodušší framework, který se uživatel snadno učí. Jedná se o rozšíření prohlížeče pro Mozilla Firefox a Google Chrome. Pro práci s IDE nemusíte mít žádné zkušenosti s programovacím jazykem. Umožňuje přístup k záznamu a přehrávání, při kterém můžete zaznamenat své testovací kroky a exportovat je do preferovaného programovacího jazyka. Tyto exportované testy lze poté použít v Selenium WebDriver (viz bod 3 níže). Obvykle se používá k vytváření testovacích prototypů.

2. **Selenium RC**

   Je v zásadě server, který umožňuje uživatelům vytvářet testovací skripty v kterémkoli z podporovaných jazyků. Tento server přidává Selenium příkazy do prohlížeče pomocí JavaSciptových příkazů. Na rozdíl od Selenium IDE nemá funkci nahrávání a přehrávání. První verze byla známá jako Selenium 1.0. Tato verze přinesla koncept paralelního a vzdáleného provádění. Měla mnoho omezení, méně realistické interakce s prohlížečem, nekonzistentní výsledky a jiné nedostatky, které později vedly k jeho opuštění a vývoji Selenium WebDriver (viz bod 3 níže).

3. **Selenium WebDriver**

   Překonává omezení Selenium RC. Na rozdíl od Selenium RC nezávisí na JavaScriptu a nemusí používat Selenium server ke spouštění testů, protože volá každý prohlížeč sám o sobě. Má rychlejší dobu provádění ve srovnání s IDE a RC. Selenium RC a WebDriver byly sloučeny do

jedné jednotky známé jako Selenium WebDriver 2.0. Postupem času byl Selenium WebDriver vylepšen o další funkce a nyní máme Selenium 4.0, které je široce používáno.

4. **Selenium Grid**

   Používá se spolu se Selenium RC k provádění paralelních testů napříč různými prohlížeči a stroji. Uživatelé mohou spouštět simultánní testy ve více prostředích současně, čímž šetří spoustu času. Implementuje koncept uzlů rozbočovače, kde každý uzel přijímá příkazy z centrálně umístěného rozbočovače.

4.5 Resharper

Resharper [5] je nástroj, který rozšiřuje možnosti Visual Studia. Analyzuje kvalitu kódu, takže uživatel hned ví, zda je třeba něco opravit. Eliminuje chyby tím, že rychle nabízí možnosti, jak opravit aktuální chybu. Umožňuje bezpečně měnit kód, na který něco odkazuje, díky automatickému refaktorování. Nabízí také rychlé procházení a hledání skrze celý projekt. Poskytuje mnoho funkcí, které jsou nápomocné při úpravách kódu a také k dodržování formátu funkcí a stylu kódu pro daný jazyk.

4.6 Nástroje firmy Atlassian

Atlassian je australská společnost poskytující podnikový software pro vývojáře a týmy. Během mé praxe jsem používal tyto čtyři nástroje:

1. **Sourcetree** [6] je multiplatformní aplikace pro práci s distribuovanými verzovacími systémy Git a Mercurial. Umožňuje pracovat s oběma verzovacími systémy v rámci jediného okna, nabízí snadné napojení na uživatelské účty služeb GitHub, Bitbucket nebo Kiln. Stejně tak pracuje se Subversion servery. Součástí jsou nástroje pro správu všech vašich repozitářů, ať už hostovaných nebo lokálních. Díky podpoře tzv. tabů lze pracovat s více repozitáři najednou.

2. **Bamboo** [7] je nástroj pro nepřetržitou integraci (continuous integration - CI). CI je proces, který se při provádění změn ve zdrojovém kódu softwaru automaticky vytvoří, otestuje a nasadí. S CI lze změny otestovat během několika minut od provedení, a to i v komplexní aplikaci s mnoha servery a komponentami.

3. **Jira** [8] je světovou jedničkou mezi bug a issue tracking systémy. Software je speciálně navržen pro účinné týmové plánování, zadávání úloh, workflow management, evidenci práce, tvorbu analýz a notifikací. S tímto nástrojem můžete procesy řídit snáze, rychleji a efektivněji.

4. **Bitbucket** [9] je webová služba firmy Atlassian podporující vývoj softwaru při používání verzovacích nástrojů Git a Mercurial. Poskytuje mnoho funkcí, které umožňují lepší spolupráci při vývoji. Umožňuje řízení přístupu, pracovního toku, pull request s vloženým komentováním pro spolupráci při kontrole kódu, sestavení aktuálního kódu a integraci s Jira software, díky kterému jde vytvářet větve (branches) přímo z Jira issues.

Kapitola 5

Návrh a implementace testů

5.1 Testovaná uživatelská rozhraní

Před návrhem se tester musí dobře seznámit s každým uživatelským rozhraním, aby znal všechny jeho funkcionality. Díky výše uvedeným testům (Manuální testy, Systémové testy pomocí robotické ruky, Automatické testy robota) a diskusím s mými zkušenými kolegy jsem získal potřebné vědomosti o produktu.

Abych měl ještě větší přehled o funkcionalitě daných uživatelských rozhraní a mohl si je kdykoli vyzkoušet, měl jsem naistalován SafeQ ve virtuálním prostředí. Díky tomu jsem při vytváření scénářů zkoušel jak pozitivní, tak negativní průběh testu. V pozitivních scénářích se testuje běžné užití, kdy výstupy a akce splňují očekávání uživatele. Naopak u negativních scénářů dochází k nestandardním postupům. Například uživatel vloží nevhodný datový typ do vstupního parametru, a z toho důvodů následně software zobrazí chybové hlášení.

Důležité je také dodržovat atomicitu. To znamená, že test by měl být co nejjednodušší, zaměřený na konkrétní část uživatelského rozhraní, aby vždy bylo jasné, co nefunguje, pokud test bude neúspěšný. Stejné pravidlo platí pro kroky v testu.

Další důležitou částí při návrhu je mockování. To znamená, že dochází k nahrazení reálného objektu za jeho testovací imitaci, která neprovádí žádnou funkcionalitu a jen se tváří jako původní objekt. Předstírá chování, které potřebujeme kvůli testování. Více informací ohledně mockování se nachází v kapitole 5.5.

Během praxe jsem vytvářel automatické testy pro čtyři způsoby zobrazení uživatelského rozhraní na tiskárně:

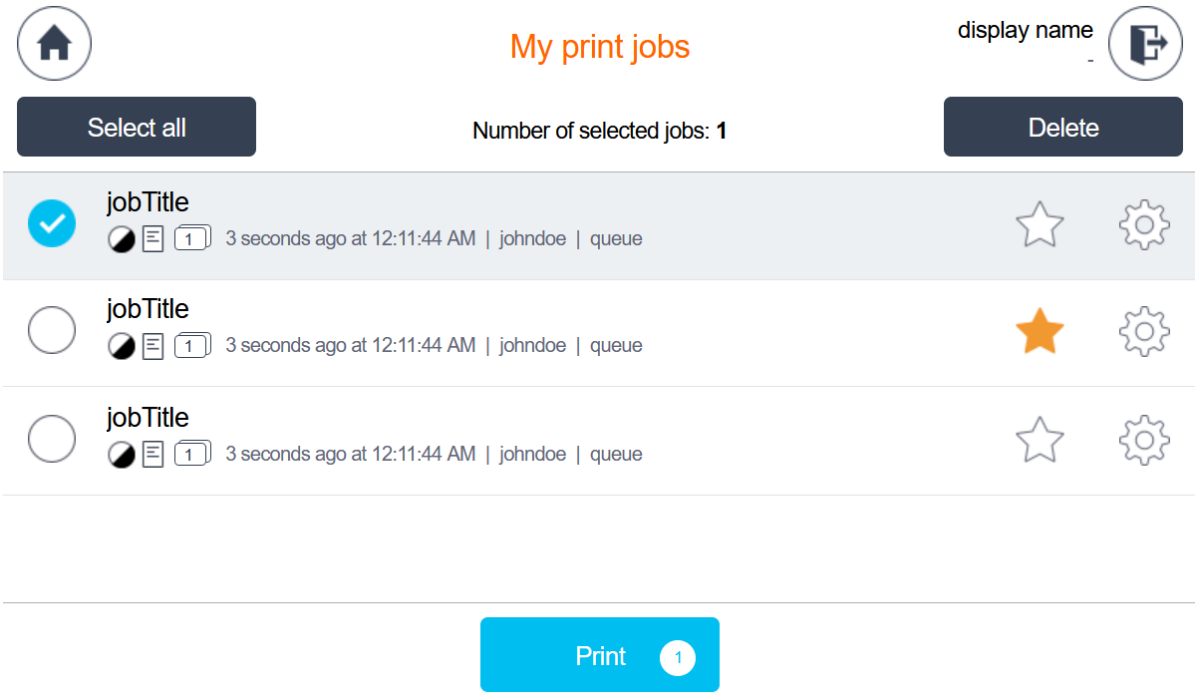
1. **Zobrazení tiskových úloh Waiting – Printed – Favorite: Záložka ve stavu vytištěné**

   V tomto uživatelském rozhraní jsou zobrazené všechny vytištěné úlohy. Každá úloha má zobrazené informace o tom, jak a kdy byla vytištěná, a jde s ní nadále pracovat (označit jako oblíbenou, vytisknout znovu nebo ji smazat). Tlačítka nahoru a dolů poskytují uživateli mož-

nost si prohlédnout všechny vytištěné úlohy. Pro toto rozhraní jsem vytvořil 26 testů, které ověřují správnost zobrazení a umístění ovládacích tlačítek, vytištěných úloh a textů. Následně testují funkcionalitu: zobrazení úlohy po prvním vytisknutí, smazání úlohy, opětovné vytisknutí, označení úlohy jako oblíbená, označení všech úloh, možnost se posunout nahoru a dolů.

2. **Zobrazení tiskových úloh All-in-one**

   Toto uživatelské rozhraní slouží pro zobrazení tiskových úloh, kde jsou čekající, vytištěné a označené jako oblíbené zobrazeny na jedné stránce. V podstatě toto uživatelské rozhraní spojuje tři jednotlivé záložky, které obsahovaly úlohy čekající, vytištěné a oblíbené, do jedné. Toto rozhraní bylo pokryté sadou testů, které testují správnost zobrazení a umístění ovládacích tlačítek, čekajících úloh, vytištěných úloh, oblíbených úloh a textů. Následně ověřují i funkcionalitu, kdy se po přihlášení uživateli zobrazí jeho úlohy, a po vytistknutí jsou pro něj stále viditelné, i když jsou již vytištěné. Uživatel může označit vytištěné a zaslané úlohy jako oblíbené, může označit úlohy jednotlivě nebo všechny najednou a následně je vymazat nebo vytisknout, může se odhlásit, posunout na této stránce nahoru a dolů. Na níže uvedeném obrázku 5.1 vidíme zobrazení uživatelského rozhraní na tiskárně: Zobrazení tiskových úloh All-in-one.



Obrázek 5.1: Stránka AllInOne

3. **Zobrazení skenovacích možností uživatele**

   Toto uživatelské rozhraní poskytuje uživateli různé možnosti skenování. Díky možnosti vytvoření skenovacího postupu, uživatel může mít více skenovacích scénářů, které mu umožňují skenování do emailu, domovské složky nebo předem určeného umístění. Také umožňují detailní nastavení skenu, kde uživatel zadává potřebné parametry a údaje pro očekávaný výstupní soubor. Já jsem toto uživatelské rozhraní rozdělil na dvě části, které byly testovány zvlášť. První byla stránka se zobrazenými skenovacími workflows. Testováno bylo: správné zobrazení stránky, posouvání nahoru a dolů na stránce, ověření přístupu ke skenování, funkčnost skenovacích workflow a jejich nastavení. Druhá část byla nastavení skenování a parametrů workflow, kde bylo testováno: zobrazení nastavení, funkčnost posouvání nahoru a dolů na této stránce a ve výběrů možností, funkčnost tlačítek zpět a odhlášení, ověření nevyhovujících parametrů, ověření chybových oznámení, ověření funkcionality nastavení skenu.

4. **Zobrazení pokročilých skenovacích možností uživatele**

   Jedná se o rozšíření uživatelského rozhraní zobrazení skenovacích možností uživatele (bod 3). Poskytuje specifické rozšířené možnosti nastavení skenu jen pro zařízení značky Konica Minolta. Nastavení skenu v tomto rozhraní obsahuje dalších 11 možností, takových jak: Scan size, Page settings, Original type, Background removal, Density, Sharpness, Blank page removal, Resolution, Sides, Color a Filetype. Testy byly stejné jako ve výše zmíněném uživatelském rozhraní, akorát rozšířené o specifické testy pro další možnosti nastavení skenu.

5.2 Vytváření scénářů

Každou sadu testů na začátku souboru jsem pojmenoval, a následně vytvořil krátký popis toho, co se v daném souboru bude testovat – co vše testy pokryjí. Většinou pak následuje za touto částí klíčové slovo Background, které vytvoří před každým spuštěním testu nastavení potřebné pro všechny testy. Jak můžeme vidět ve výpisu kódu 5.1, za klíčovým slovem Feature v prvním řádku určím název souboru. V následujících řádcích je pak již popis, co všechno bude testováno. Dále je pak klíčové slovo Background, které určuje v krocích Given potřebné nastavení. V tomto případě před každým testem bude naistalován terminál se zobrazením tiskových úloh All-in-one a autorizací přes pin.

```
Feature: Job list with All in one layout
    As a tester, I want to have operations in ALL IN ONE
    application layout covered with UI tests
    so I can ensure its functionality
```

```
Background:
    Given terminal is installed with 'Pin' authentication
        And terminal is installed with Printing application
            layout 'AllInOne'
```

Listing 5.1: Feature and Background

Při vytváření testovacího scénáře u testu s více vstupními parametry (Scenario Outline), se za klíčovým slovem Scenario vždy nachází výstižný název, který by měl jednoznačně určit, co se v daném scénáři bude testovat. Pak se nachází další potřebná klíčová slova pro vytvoření testovacího scénáře (Given, When, Then). Každý krok by měl být co nejjednodušší a název dobře srozumitelný. Na následujícím výpisu kódu 5.2 vidíme vytvořený test s více vstupními parametry pro ověření, že uživatel může označit různý počet tiskových úloh. V části Given vidíme dva kroky, které vytvoří potřebný počáteční stav testu. V tomto případě je to stav, kdy má uživatel zaslané úlohy a je přihlášen. Dále pak vidíme krok When, ve kterém můžeme zaznamenat akci - konkrétně výběr úlohy uživatelem. V části Then je následně ověřeno, zda označené úlohy jsou zobrazené správně. Tlačítka a texty, které se dle dané funkcionality uživatelského rozhraní mají zobrazit, se zobrazí tak, jak mají. Testy jsou prováděny i pro případy, kdy je více stránek s úlohami. Podle velikosti obrazovky terminálu určujeme počet úloh na stránku, ale pro potřeby testů používáme standardní velikost, kde se vejdou čtyři úlohy. Pod scénářem v části examples jsou napsané hodnoty parametrů. Každý řádek je samostatný test. Vidíme, že proběhne pět samostatných testů: uživatel vybere jednu úlohu z více úloh, uživatel vybere několik úloh z více úloh, uživatel vybere všechny úlohy na stránce, vybere několik úloh na jedné stránce z více stránek a uživatel vybere úlohy z více stránek.

```
Scenario Outline: User can select jobs
    Given user sends '<jobCount>' jobs to SafeQ
        And user is successfully logged in
    When user select '<jobsSelected>' jobs
    Then '<jobsSelected>' jobs are selected
        And select all button appears
        And delete button appears
        And text Number of selected jobs: '<jobsSelected>' appear
        And print button have number '<jobsSelected>' inside
```

```
Examples:
    | jobCount | jobsSelected |
    | 2        | 1            |
    | 4        | 3            |
    | 4        | 4            |
    | 8        | 2            |
    | 8        | 5            |
```

Listing 5.2: Test na ověření správného označování úloh

5.3 Implementace

Pro lepší přehlednost implementace kroků, které jsou použity v testech daného uživatelského rozhraní, jsou uložené v samostatném souboru. Je zde pár rozhraní, které souvisejí se seznamem úloh a jsou již pokryté testy. Aby se předešlo duplicitě v implementaci, mají tyto testy společnou složku, kde mají implementované kroky. Pří vytváření testů pro uživatelské rozhraní: Zobrazení tiskových úloh All-in-one a Zobrazení tiskových úloh Waiting – Printed – Favorite: Záložka ve stavu vytištěné, jsem vždy po vytvoření scénáře kontroloval, jestli už nějaký krok ze scénáře se stejnou funkcionalitou není implementován. Díky opakovaně použitým krokům jsem zajistil vyšší udržitelnost a škálovatelnost.

Ve výpisu kódu 5.3 vidíme použité konstrukce [Binding] a následně specifikování, ve kterých souborech s definicemi testů můžeme kroky implementované ve třídě *CommonLayoutSteps* použít. Před každou implementací kroku jsou hranaté závorky, které vždy začínají klíčovým slovem, jež určí, o jaký krok se jedná (Given, When, Then). Následně v kulatých závorkách za znakem `@` je již název konkrétního kroku. V případě, že se krok používá i s jiným klíčovým slovem, lze přidat ještě další řádek, kde se pouze vymění klíčové slovo. Ve výpisu kódu můžeme vidět, že takto je nadefinován krok "*select '(.*)' jobs*", který je používán jak s klíčovým slovem Given, tak When. V další části je již implementace kroku v jazyce C#, která většinou volá potřebné funkce aktuálně zobrazené stránky. V tomto případě je to stránka *JobListPage* a její funkce *SelectJobsFromTop(selectedJobsCount)*.

```
namespace YSoft.Sqta.IntegrationTests.Specs.JobListFeature.Steps
{
    [Binding]
    [Scope(Feature = "Print status")]
    [Scope(Feature = "Quick Actions")]
    [Scope(Feature = "Print Cancelling On Logout")]
    [Scope(Feature = "Job list with All in one layout")]
    [Scope(Feature = "Job list with waiting layout - waiting tab")]
```

```
    internal class CommonLayoutSteps : StepsBase
    {
        [Given(@"user select '(.*)' jobs")]
        [When(@"user select '(.*)' jobs")]
        public void WhenUserSelectJobs(int selectedJobsCount)
        {
            var currentPage = GetCurrentPage<JobListPage>();
            var selectedJobsIndices = currentPage.SelectJobsFromTop(
                selectedJobsCount);
            JobListHelper.SelectedJobCount = selectedJobsCount;
            JobListHelper.SelectedJobsIndices = selectedJobsIndices;
        }
    }
}
```

Listing 5.3: Soubor s implementovanými kroky v třídě CommonLayoutSteps

5.4 Rozšiřování funkcionality

Občas jsem narazil na implementovaný krok, který měl podobnou funkcionalitu, akorát byl omezený, a tudíž ho nešlo použít v mém scénáři. V případě takové implementace kroku je zapotřebí ji rozšířit, aby byla co nejobecnější. Ve výpisu zdrojového kódu 5.4 vidíme původní funkci, která je použitá v implementaci kroku *"user select '<jobsSelected>' jobs"* ve scénáři výpisu zdrojového kódu 5.2. Tato funkce, jak můžeme vidět, označí a následně spočítá úlohy na jedné stránce. Můžeme si všimnout, že ve výše uvedeném scénáři má tato funkce označit a spočítat daný počet úloh i na více stránkách. Proto jsem musel rozšířit funkcionalitu této funkce, aby ji bylo možné použít, jak to vyžaduje výše uvedený scénář. Ve výpisu zdrojového kódu 5.5 je zobrazená již vyhovující implementace funkce *SelectJobsFromTop*, která je schopná označit a spočítat úlohy na více stránkách.

```
    public class JobListPage : PageBase, INavigablePage
    {
        const string jobIsSelectedClass = "tf-list-item_selected";
        const string jobIdPrefix = "job-";

        public ReadOnlyCollection<IWebElement> JobList => Driver.FindElements(By.
            XPath("//li[contains(@id, 'job-')]"));
```

```
    public List<int> SelectJobsFromTop(int selectedJobsCount)
    {
        for (var i = 0; i < selectedJobsCount; i++)
        {
            JobList[i].Click();
        }

        var selectedElements = Driver.FindElement(By.Id("js-tf-job-list")).
            FindElements(By.ClassName(jobIsSelectedClass));
        var parsedIDs = selectedElements
            .Select(webElement => Int32.Parse(webElement.GetAttribute("id").
                Replace(jobIdPrefix, ""))).ToList();

        return parsedIDs;
    }
}
```

Listing 5.4: Původní implementace funkce SelectJobsFromTop()

Jak můžeme vidět, v níže uvedeném výpisu zdrojového kódu 5.5 je funkce, která je specifická pro uživatelské rozhraní, které obsahuje seznám úloh. Z tohoto důvodu pro lepší přehlednost je vytvořená v třídě *JobListPage*, která je rozšířením třídy *PageBase*. Na začátku této třídy jsou vytvořené konstanty, které se používají ve více funkcích. Jednotlivé elementy této stránky (Zobrazení tiskových úloh All-in-one), které v testech využíváme, jsou v dané třídě deklarovány jako proměnné typu IWebElement. Dále pak pomocí dvou z hlavních funkcí Selenium WebDriver (*Findelement* a *Findelements*) volaných z třídy Driver, která je typu FirefoxDriver a je zděděná ze třídy PageBase, můžeme najít potřebné prvky na aktuální stránce a následně s nimi i pracovat. Tyto dvě funkce můžou na webové stránce vyhledávat elementy pomocí jména, jména třídy, id, tagu nebo Xpath. V níže vypsaném zdrojovém kódu vidíme použití obou funkcí, kde *Findelements* odkazuje na seznam úloh na stránce a *Findelement* odkazuje na tlačítko posunout dolů.

```
public class JobListPage : PageBase, INavigablePage
{
    const string jobIsSelectedClass = "tf-list-item_selected";
    const string jobIdPrefix = "job-";
    const int maxJobsOnPage = 4;
```

```
    public ReadOnlyCollection<IWebElement> JobList => Driver.FindElements(By.
        XPath("//li[contains(@id, 'job-')]"));
    public IWebElement ScrollDownButton => Driver.FindElement(By.Id("
        navigationPanel-pageDown"));
}
```

Listing 5.5: Aktuální implementace funkce SelectJobsFromTop() - 1. část

Následně v níže uvedeném kódu 5.6 je již rozšířená funkce *SelectJobsFromTop()*, která vrací seznam počtu úloh na jednotlivých stránkách. Funkce vytvoří seznam (*parsedIDs*), do kterého bude průběžně ukládat počet označených úloh na jednotlivých stránkách. Následně v cyklu *for* projde jenom potřebný počet stránek, pro úspěšné označení a spočítání všech úloh, které jsou dané parametrem funkce *selectedJobsCount*. Dále pak do proměnné *jobsToGo* uložíme počet úloh, které se musí ještě označit a spočítat. Díky ternárnímu podmíněnému operátoru přiřadíme hodnotu do parametru *jobCountToSelect* dle toho, zda počet úloh na označení je větší nebo rovný maximálnímu počtu na aktuální stránce (maximální počet při těchto testech na stránce je 4). Pokud počet úloh na označení je menší než maximální počet, tak označí daný počet dle parametru *jobsToGo*, jinak označí vše. V dalším cyklu *for* pomocí volané funkce *Click()* na jednotlivých prvcích kolekce *JobList*, označíme počet úloh dle výše zmíněného parametru *jobCountToSelect*. Další proměnná *selectedElements* díky funkcím *Findelement* a *Findelements* odkazuje na aktuálně označené úlohy na stránce. Následně přidáme do již výše zmíněného seznamu *parsedIDs* seznam aktuálně označených úloh. Následně se díky funkci *click* na tlačítku posunout dolů (*ScrollDownButton*) posune o další stránku níž, dokud není označený stejný počet úloh jako uvádí parametr funkce (selectedJobsCount).

```
public class JobListPage : PageBase, INavigablePage
{
    public List<int> SelectJobsFromTop(int selectedJobsCount)
    {
        var parsedIDs = new List<int>();

        for (var i = 0; i <= (selectedJobsCount - 1) / maxJobsOnPage; i++)
        {
            var jobsToGo = selectedJobsCount - parsedIDs.Count();
            var jobCountToSelect = jobsToGo >= maxJobsOnPage ? maxJobsOnPage :
                jobsToGo;

            for (var j = 0; j < jobCountToSelect; j++)
            {
                JobList[j].Click();
            }
```

```
            var selectedElements = Driver.FindElement(By.Id("js-tf-job-list")).
                FindElements(By.ClassName(jobIsSelectedClass));
            parsedIDs.AddRange(selectedElements
                .Select(webElement => int.Parse(webElement.GetAttribute("id").
                    Replace(jobIdPrefix, ""))).ToList());

            if (parsedIDs.Count() == selectedJobsCount)
                break;

            ScrollDownButton.Click();
        }

        return parsedIDs;
    }
}
```

Listing 5.6: Aktuální implementace funkce SelectJobsFromTop() - 2. část

5.5 Mockování

Při vytváření testů pro různá uživatelská rozhraní komponenty ETUI jsem potřeboval vytvořit imitaci jiné komponenty (Mock), se kterou testovaná komponenta komunikuje. Důvodem vytvoření takové imitace je zajištění, že při komunikaci s komponentou, která není testována, vždy dojde k předem určenému výsledku. Tímto způsobem zajistíme potřebné podmínky pro námi vytvářené testy. Při vytváření testů pro uživatelské rozhraní Zobrazení skenovacích možností uživatele, jsem potřeboval otestovat správné zobrazení hlášení chybových vstupů skenovacích parametrů. V tomto případě mě nezajímala funkcionalita, ale jenom správné zobrazení, a proto jsem potřeboval vytvořit mock pro vstupní parametry skenu.

Ve výpisu zdrojového kódu 5.7 vidíme část vytvořené funkce *FakeIncompatibleInputs*, která přijímá seznam typů chybových hlášení (*typesOfParameter*). Celkový počet různých typů hlášení je 4. V níže vypsaném kódu vidíme jenom jeden z nich (*ScanValidationErrorType.ParameterNotDate*). Tato funkce na začátku vytvoří proměnnou *invalidInput*, která bude průběžně zaznamenávat chybová hlášení. Dále pak proměnnou *orderOfParameters*, která slouží jako ID při vytváření parametru, a po každém vytvoření parametru se zvýší o 1. V cyklu *foreach* funkce projde celý seznam *typesOfParameter* a pro každý typ hlášení v seznamu vytvoří daný parametr. Následně vytvoří a přidá do nového seznamu (*invalidInput*) specifické chybové hlášení pro tento parametr. Jedním z prvních kroků při vytváření mocku je určit, které volání se má konfigurovat. Díky knihovně FakeItEasy a

její třídě *A* s funkcí *CallTo* můžeme určit, jak se bude imitace objektu chovat. V mém případě imitace objektu *scanWorkflowValidator* a jeho funkce *Validate*, která ověřuje vstupy parametrů. Protože testuji jenom zobrazení chybových hlášení, tak ve funkci *Validate* vložím jako parametr *A<ScanWorkflow>._* , který díky "._"říká, že jakékoli vstupy budou v tomto workflowu, tak má zavolat funkci *Returns*. Tato funkce již vrací námi určený seznam chybových hlášení *invalidInput*.

```
public IFakeCalls FakeIncompatibleInputs(List<ScanValidationErrorType>
    typesOfParameter)
{
    var invalidInput = new List<ScanValidationError>();
    int orderOfParameters = 1;
    foreach (var parameter in typesOfParameter)
    {
        switch (parameter)
        {
            case ScanValidationErrorType.ParameterNotDate:
            {
                var parameterWithDateType = new ScanParameter(
                    orderOfParameters, "name", "label", false,
                    "defaultvalue", true, ScanParameterDataType.Date);

                invalidInput.Add(new ScanValidationError(
                    parameterWithDateType,
                    ScanValidationErrorType.ParameterNotDate));
                break;
            }
            default:
                throw new ArgumentOutOfRangeException();
        }
        orderOfParameters++;
    }

    A.CallTo(() => scanWorkflowValidator.Validate(A<ScanWorkflow>._)).
        Returns(invalidInput);
    return this;
}
```

Listing 5.7: Mockování vstupů parametrů skenu

Kapitola 6

Zhodnocení dosažených výsledků

6.1 Znalosti a dovednosti získané v průběhu studia uplatněné v průběhu odborné praxe

V průběhu odborné praxe jsem využil celou řadu znalostí a dovedností, které jsem nabyl během studia na Vysoké škole báňské – Technické univerzitě Ostrava. Rád bych vyzdvihl některé z nich.

Mezi nejcennější znalosti bych osobně zařadil ty získané v předmětech, které se věnují především programování a programovacím jazykům. Zejména Programování I, Programování II a Programovací Jazyky I. Tyto předměty daly solidní základ mým programátorským schopnostem. Jelikož většinu času mé praxe jsem programoval automatizované testy v jazyce C#, za nejpřínosnější pokládám znalosti a dovednosti nabyté v předmětu Programovací jazyky II, ve kterém jsem se naučil základní techniky programovacího jazyka C#. Díky získaným vědomostem z předmětů Algoritmy I a Algoritmy II jsem byl schopen snadněji a efektivněji řešit úkoly, se kterými jsem pracoval. Z předmětu Úvod do softwarového inženýrství jsem si v praxi vyzkoušel, jak funguje agilní metodika inkrementálního vývoje Scrum.

6.2 Znalosti a dovednosti scházející studentovi v průběhu odborné praxe

V průběhu odborné praxe mi chyběly hlavně znalosti z oblasti testování softwaru, s kterými jsem se bohužel na bakalářském studiu detailněji nesetkal. Kvůli tomu jsem se musel seznámit s celým procesem testování softwaru a také používanými frameworky a nástroji ve firmě. Tyto nástroje a frameworky jsou popsané v kapitole 4 Použité technologie. Další znalosti, které mi chyběly při vykonávání praxe, byly vědomosti o verzovacích systémech, které nevyhnutelně patří k vývoji softwaru. V této oblasti jsem se seznámil a následně pracoval s nástrojem Sourcetree, který nabízí grafické rozhraní nad Gitem a zjednodušuje práci s jeho repozitáři.

6.3 Závěr

Absolvování individuální odborné praxe ve firmě Y Soft Corporation, a.s. hodnotím jako skvělou zkušenost, díky které jsem se naučil pracovat v týmu a měl možnost nabýt nové vědomosti. Tato praxe mi také poskytla možnost vyzkoušet více způsobů testování softwaru, takže jsem získal široké spektrum znalostí v této oblasti a také potřebné informace o hlavním produktu firmy, kterým je YSoft SafeQ. Nabyté znalosti mi pak umožnily se plně věnovat svému hlavnímu úkolu, a to automatizovaným testům.

Výsledkem mé odborné praxe je vytvoření 96 automatizovaných UI testů pro terminálovou část YSoft SafeQ. Tyto testy pokrývají čtyři uživatelská rozhraní, které jsou popsané v kapitole 5 Návrh a implementace testů. Jsem velice rád, že jsem mohl vypracovat bakalářskou práci formou praxe, a tím si vyzkoušet práci ve firmě, absolvovat standardní pohovor a řešit reálné problémy, které mohou nastat během vykonávání práce. Naučil jsem se mnoho a vím, že takto získané dovednosti jsou velkým přínosem pro mé budoucí profesní uplatnění.

Literatura

[1] Y Soft [online]. [cit. 2021-03-11]. Dostupné z: https://www.ysoft.com/cs

[2] SpecFlow - Binding Business Requirements to .NET Code [online]. [cit. 2021-03-11]. Dostupné z: https://specflow.org/

[3] Gherkin Reference - Cucumber Documentation [online]. [cit. 2021-03-11]. Dostupné z: https://cucumber.io/docs/gherkin/reference/

[4] SeleniumHQ Browser Automation [online]. [cit. 2021-03-11]. Dostupné z: https://www.selenium.dev/

[5] ReSharper: The Visual Studio Extension for .NET Developers by JetBrains [online]. [cit. 2021-03-11]. Dostupné z: https://www.jetbrains.com/resharper/

[6] Sourcetree - A free Git & Mercurial client [online]. [cit. 2021-03-11]. Dostupné z: https://www.atlassian.com/software/sourcetree

[7] Bamboo Continuous Integration and Deployment Build Server [online]. [cit. 2021-03-11]. Dostupné z: https://www.atlassian.com/software/bamboo

[8] Jira Software [online]. [cit. 2021-03-11]. Dostupné z: https://www.atlassian.com/software/jira

[9] Atlassian Bitbucket Git Code Management Tool for Teams [online]. [cit. 2021-03-11]. Dostupné z: https://www.atlassian.com/software/bitbucket